**Question 1**

Here are the rules of the grammar:

(1) L 🡪 Ra

(2) | Qba

(3) R 🡪 aba

(4) | caba

(5) | Rbc

(6) Q 🡪 bbc

(7) | bc

Here we can see that rule 5 has a left recursion. So, it is not suitable for top down predictive parsing such as LL1 grammars. So first, we need to eliminate left recursion.

R 🡪 abaR’ | cabaR’

R’ 🡪 bcR’ |

Then we need to perform left factoring on Rule 6-7.

Q 🡪 bX

X 🡪 bc | c

Hence the new set of grammar will be the following and its first and follow sets.

|  |  |  |
| --- | --- | --- |
| **Rules** | **First Sets** | **Follow Sets** |
| (1) L 🡪 Ra | a, c | $ |
| (2) | Qba | b | $ |
| (3) R 🡪 abaR’ | a | a |
| (4) | cabaR’ | c | a |
| (5) R’ 🡪 bcR’ | b | a |
| (6) R’ 🡪 |  | a |
| (7) Q 🡪 bX | b | b |
| (8) X 🡪 bc | b | b |
| (9) | c | c | b |

So, here is the LL1 parsing table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **a** | **b** | **c** | **$** |
| **L** | 1 | 2 | 1 |  |
| **R** | 3 |  | 4 |  |
| **R’** | 6 | 5 |  |  |
| **Q** |  | 7 |  |  |
| **X** |  | 8 | 9 |  |

Hence, there is no multiple entries in a single cell referring to the fact that, we can determine which production needs to be chosen based on next input. So, the grammar is LL1 compliant.

**Question 2**

1. A 🡪 Ba
2. B 🡪 dab
3. | Cb
4. C 🡪 cB
5. 🡪 Ac

From this grammar, we can see that,

A 🡪 Ba or, A 🡪 Cba or, A 🡪Acba

Hence, this grammar contains left recursion.

As it contains left recursion, it is not a LL1 grammar. However, we can also prove it from another perspective.

The *first* set of rule 4 is c

The *first* set of rule 5 is c, d

Hence, we can see *First*(Rule 4) *First*(Rule 5) . Thus, it is not a LL1 grammar.

So, we need to remove left recursion first.

A 🡪 Ba or A 🡪 daba | Cba or A 🡪 daba | cBba | Acba

can be transformed to

A 🡪 dabaA’ | cBbaA’

A’ 🡪 cbaA’ |

So, the final set of grammar will be,

1. A 🡪 dabaA’ | cBbaA’
2. A’ 🡪 cbaA’ |
3. B 🡪 dab | Cb
4. C 🡪 cB | Ac

**Question 3**

The preliminary intuition for this grammar is very simple. Apart from 0, all binary numbers divisible by 4 ends with 000 or 100. The leading bits can form any binary number.

So, the number is: 0 or *Any Binary Number* having 000 or 100 in the end. The regular expression is:

([0|1])+(000|100) | 0 | 00 | 000 | 100

Hence, the grammar is given below. Non-terminals are highlighted with <>.

<GOAL> 🡪 0 | <BINARY-NUMBER>000 | <BINARY-NUMBER>100

<BINARY-NUMBER> 🡪 <BIT> |

<BIT> 🡪 <BIT><BIT>

<BIT> 🡪 0 | 1